Ali Ismail

Lab 4 Write Up

Required Elements

Required

1 LCD Logic and Timing

The first step in designing the LCD is to ensure the LCD logic and timing requirements are met. The LCD has three control signals that must meet timing requirements, enable (E), read/write(R/W), and register select (RS). First and foremost, E dictates when the LCD is enabled. To control this signal, I use the /WR, /RD, and A12-A15 signals from the 8051. The logic below describes the logic:

$$E = (A15 * A14 * A13 * A12) * (!RD + !WR)$$

The address signals are ANDed together to ensure that the LCD is memory mapped to 0xF000 to 0xFFFF. Active low signals /WR and /RD indicate when there has been a read or write from the microcontroller. If either is triggered, then this indicates the possibility that the 8051 is attempting a read or write to the LCD. They are negated since they are active low, and ORed together since either requires an enabled LCD. As mentioned earlier, the LCD is memory mapped, therefore, if the 8051 does a read or write and the address is between 0xF000 and 0xFFFF, then it is attempting to talk to the LCD. The logic above reflects this scenario and will cause E to go high. Figure 1 shows the simulation of the logic above. As you can see, the enable is only trigged when a read or write occurs and in the correct memory space.

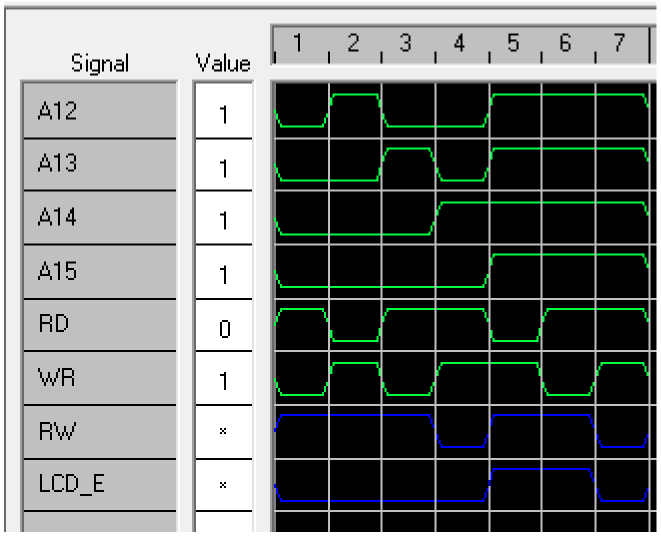


Figure 1

The next logic to consider is for R/W and RS. The logic is as follows:

R/W = A8 and RS = P1_3

The idea behind using an address as the R/W is to distinguish between a read and write and for timing purposes. In code, there are two address distinct addresses used when the 8051 does a read or write to the LCD. The read address is 0xF100, which sets R/W as 1. The write address is 0xF000, which sets R/W as 0. Take note again that A8 is connected to R/W. In addition, using a high byte address bit to trigger R/W helps meet timing. Figure 2 below shows the required timing for the LCD. Once R/W is trigged, there needs to be 40 ns (t_{as}) before the Enable goes high. In addition, R/W must stay trigged for 10 ns (t_{ah}) after E goes low. Now take a look at figure 3. This is the write cycle time for the 8051. Port 2 is valid for T_{AVWL}, before the WR is triggered. It is also held valid for at least T_{WHQX}.

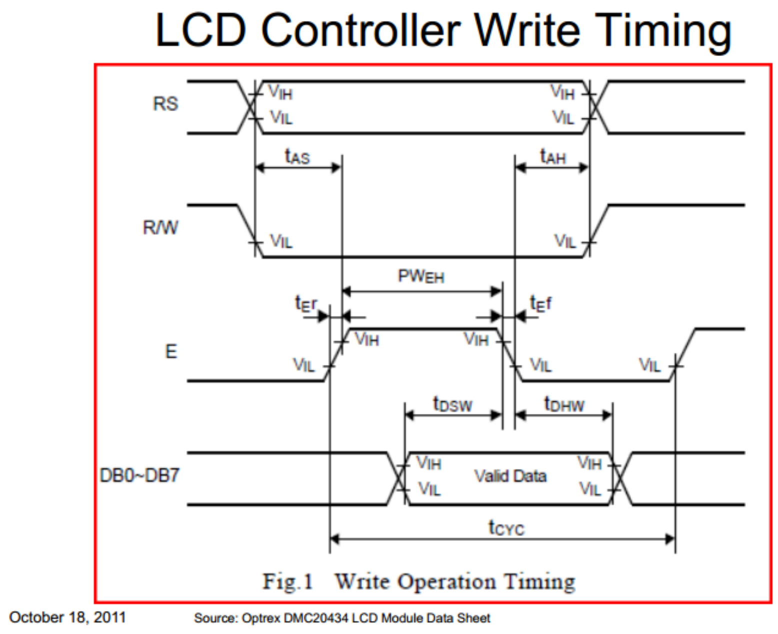

LCD Controller Write Timing
RS
R/W
E
DB0~DB7
VIH
VIL
tAS
tAH
PWEH
tEr
tEf
tDSW
tDHW
Valid Data
tCYC
Fig.1 Write Operation Timing
October 18, 2011
Source: Optrex DMC20434 LCD Module Data Sheet

Figure 2

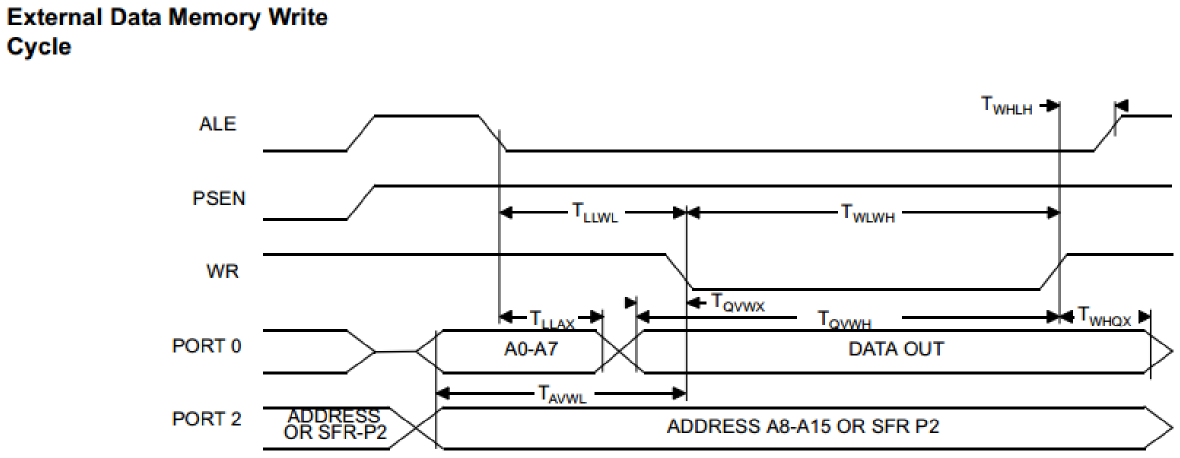


Figure 3

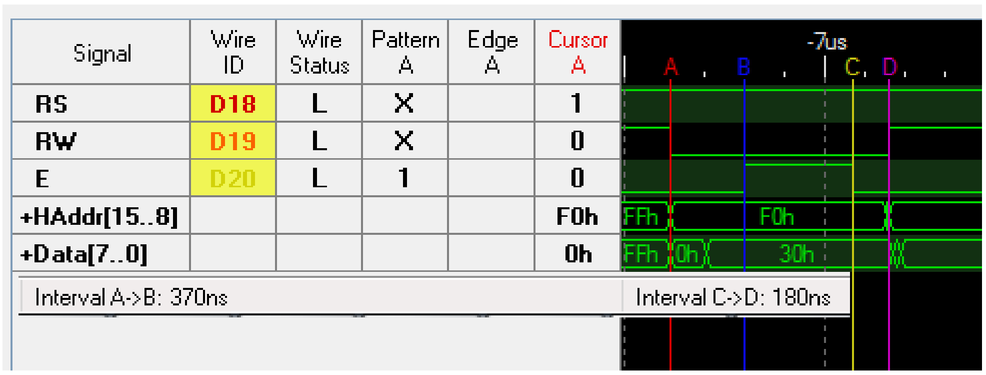
Below is the data sheet calculation for T_{AVWL}(315ns) and T_{WHQX}(30 ns). By using A8 (P2) for R/W, the setup time should be 315 ns and hold time should be 30 ns. This is more than adequate for meeting the R/W LCD requirement for a 40 ns setup time and 10 ns hold time.

$$t = 90\ ns = \frac{1}{11.0592\ MHz}$$

$$T_{AVWL} = 4t - x = 315 \text{ ns}$$

$$T_{WHQX} = .5t - x = 30\ ns$$

Figure 4 below shows the measured setup (t_{as}) and hold (t_{ah}) times for an LCD **write**. They are 370 ns and 180ns. They meet the timing requirements. Also, figure 5 shows an LCD **read**. It is clearly meets timing since R/W is always low. Also note that even though the address is in range, the LCD does not respond until E is triggered and the address is valid.

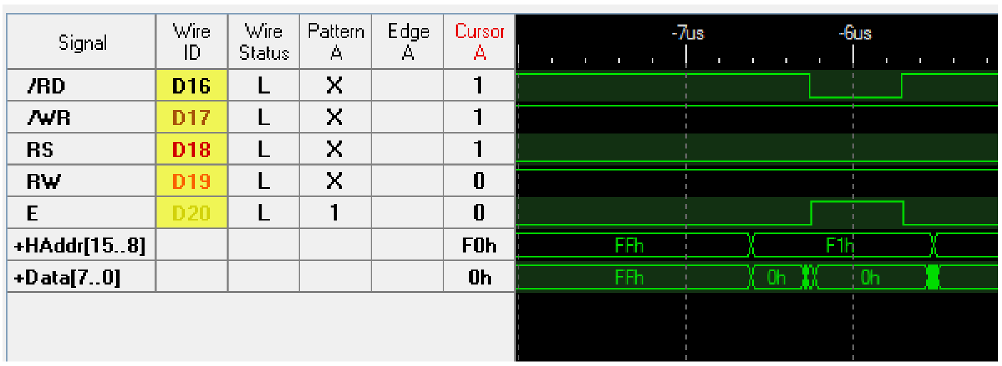


Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
RS	D18	L	X		1
RW	D19	L	X		0
E	D20	L	1		0
+HAddr[15..8]					F0h
+Data[7..0]					0h

Interval A->B: 370ns

Interval C->D: 180ns

Figure 4



Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
/RD	D16	L	X		1
/WR	D17	L	X		1
RS	D18	L	X		1
RW	D19	L	X		0
E	D20	L	1		0
+HAddr[15..8]					F0h
+Data[7..0]					0h

Figure 5

Finally, it was necessary to write a function that delays for a specified amount of time (ms). During the first three commands of the LCD initialization, the status read is not functioning, therefore, there must be a delay to allow the LCD to process the commands. Figure 6 is an oscilloscope trace for a ms delay that is called n times, depending on how many ms is needed. Figure 6 indicates that the ms function has the correct timing.

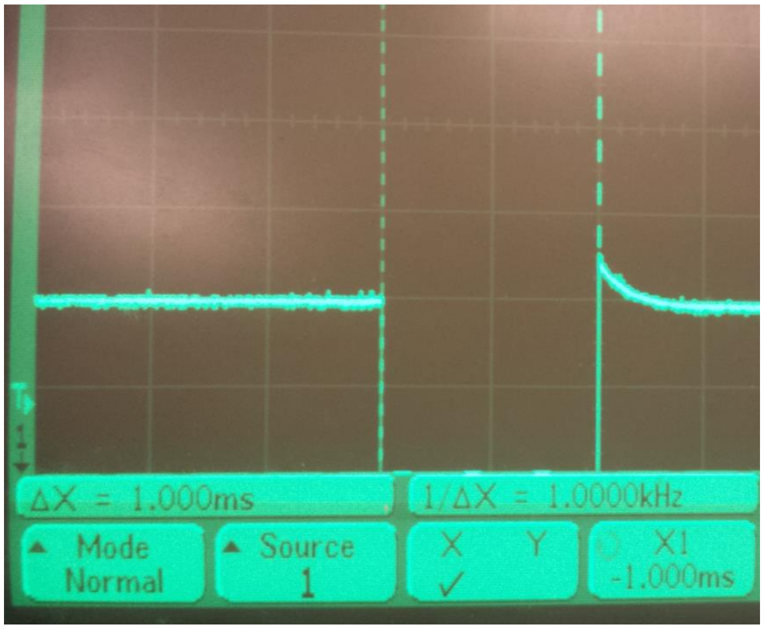
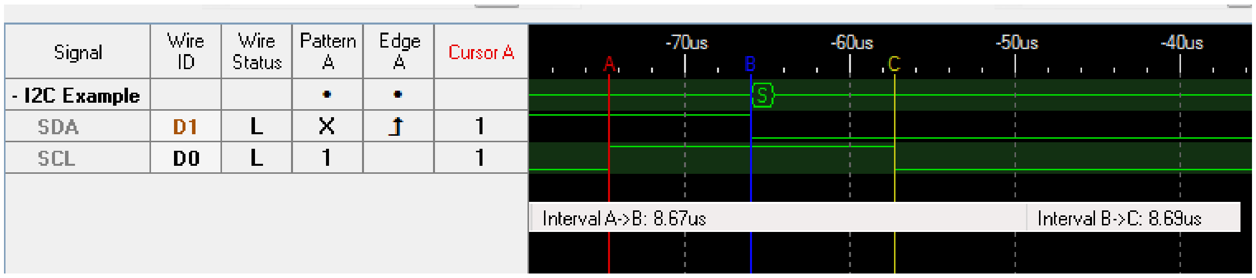


Figure 6

2 I2C EPROM Timing and Verification

The 2k EEPROM utilizes an I2C interface. I2C, in general, requires a start condition, data send/receive, and stop condition. Once these functions work correctly, they can be put together to achieve a write or read. The first part to verify was the timing and functionality of the start sequence. The start sequence is a SDA falling edge transition when the SCL is high. This can be seen in figure 7. Also, there is a setup and hold time requirement for this transaction. The EEPROM requires a start condition setup time of 0.25 us. As you can see, the measured time was 8.67 us. Also, the EEPROM requires a start condition hold time of 0.25 us and the measured time was 8.69 us. This meets the timing requirements for the start condition.



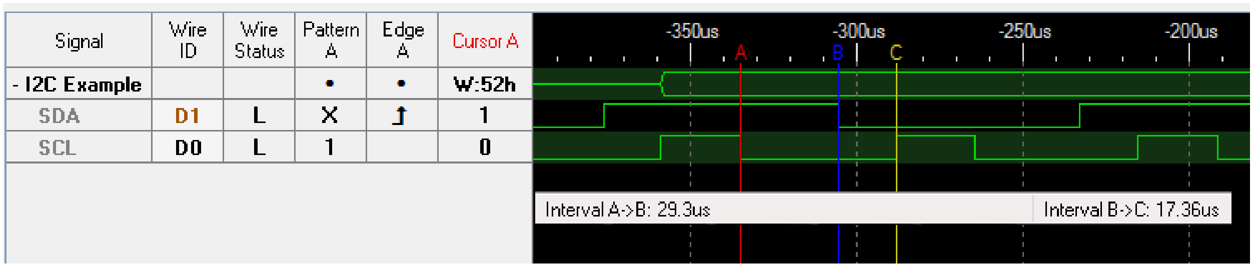
Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
- I2C Example			•	•	
SDA	D1	L	X	↥	1
SCL	D0	L	1		1

Interval A->B: 8.67us

Interval B->C: 8.69us

Figure 7: Start Condition Timing

The second part to verify was the timing and functionality of a data condition sequence. The SCL signal gives a slave a reference for timing. The EEPROM in this case is the slave and the 8051 is the master. When the master sends a clock signal after the start sequence, then the slave should prepare itself for receiving data. Figure 8 below shows the data condition during the sending of a slave address to the EEPROM. The EEPROM requires a data condition setup time of 100 ns and a hold time of 0 us. The measured data condition setup and hold times were 29.3 us and 17.36 us. The timing requirements for the data condition have been met.

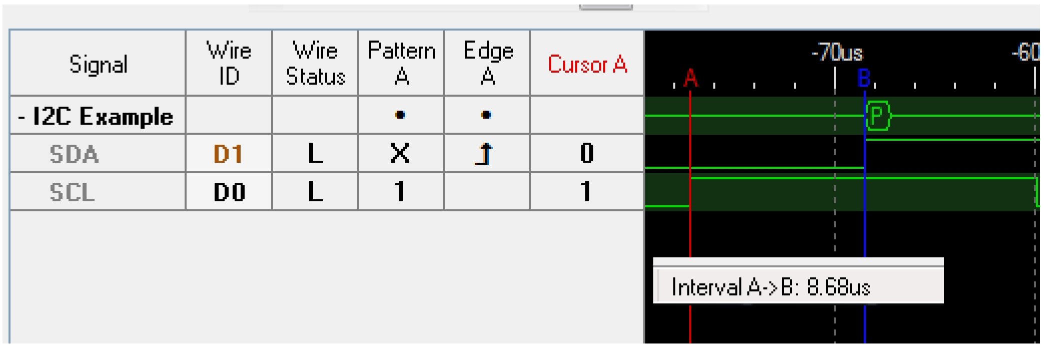


Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
- I2C Example			•	•	W:52h
SDA	D1	L	X	↥	1
SCL	D0	L	1		0

Interval A->B: 29.3us
Interval B->C: 17.36us

Figure 8: Data Condition Timing

The third part to verify was the timing and functionality of the stop sequence. The stop sequence is a SDA rising edge transition when the SCL is high. This can be seen in figure 9. The EEPROM requires a stop condition setup time of 0.25 us. As you can see, the measured time was 8.68 us. This meets the timing requirement for the stop condition.

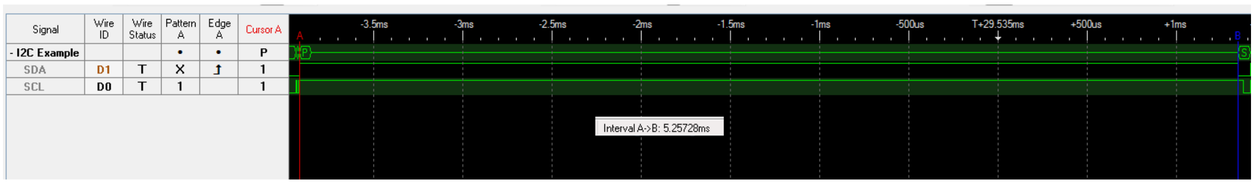


Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
- I2C Example			•	•	
SDA	D1	L	X	↥	0
SCL	D0	L	1		1

Interval A->B: 8.68us

Figure 9: Stop Condition Timing

In the event acknowledge polling is not utilized, there is a max write cycle time of 5 ms. Since there is a ms delay function, it can be used to meet the 5 ms requirement. This ensures the EEPROM has finished its write transaction and allows for another transaction. Figure 10 shows this happening. This is not the most efficient way, since the EEPROM may finish before the 5ms max time, thus, ack polling is used. The EEPROM is continually invoked until it indicates it is ready for another transaction.

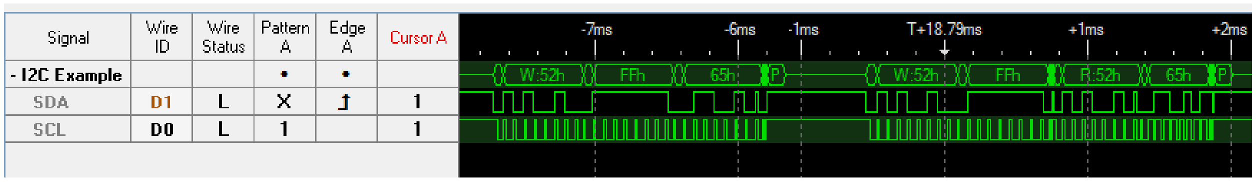


Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
- I2C Example			•	•	P
SDA	D1	T	X	↥	1
SCL	D0	T	1		1

Interval A->B: 5.25728ms

Figure 10: Write Cycle Time

Figure 11 shows a complete write and read transaction. You can see the transaction on the left is the send since there is a slave address (52h does not include R/W), word address (FFh), and data (65h) to be written. Later, there is a random read transaction. The slave and word address are defined to set the address. Next comes the slave address and the data to be read and finally a stop signal.



Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
- I2C Example			•	•	
SDA	D1	L	X	↥	1
SCL	D0	L	1		1

Figure 11: Send and receive transaction

Supplemental

1 Clock

For the clock displayed on the LCD, I used timer 0 to interrupt every 1/100th of a second. I then have a counter for each time digit, tenth of a second, second, tens of seconds, etc. I also print these count values to the LCD to create my timer. In addition, many parts of the LCD functions had to be made critical to ensure no collisions occurred between an LCD write from the timer interrupt and the main program writes to the LCD.

2 Create Custom Character

In order to allow a user to create a custom character, I ask for a valid custom code and hex values that represent each row of the 5x8 character. In order to allow the user to see their character, I printed ‘0’s for true and ‘ ‘ for false after each row was entered. This allowed the user to see the progression of their character.

3 I/O Expander

The software for the I/O expander uses an array to track the state of each pin (input or output). If the user wanted to write to a pin, there is a check done before hand to ensure that pin was an output pin. Likewise, to handle writes to a pin, the status of the pins had to be read and rewritten so the other pins were not affected. The reason for this is the I/O expander does a write to all pins and not just a single pin. Also, a push button was connected to pin 3 of the expander to cause an interrupt. The button was connected to ground, since every input pin must be set to 1 to receive data. The expander interrupts when there is an external change to the pin, thus, a change from 1 to 0 would cause an interrupt.

In the I/O expander interrupt (connected to /INT0), are counts that are controlled by same timer for the clock. The only difference is, when the I/O expander interrupt is triggered, they are printed to the LCD and zeroed. This ensures the lap times stop from zero after every lap.

4 EERESET

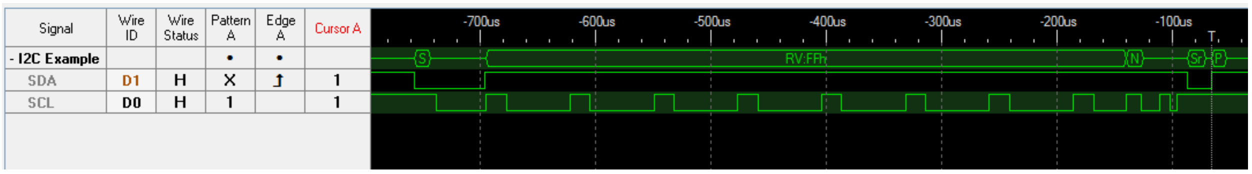
Figure 12 below shows the reset sequence sent to the EEPROM. The sequence is as follows:

Start

Send 9 1's

Start

Stop



Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
- I2C Example			•	•	
SDA	D1	H	X	↑	1
SCL	D0	H	1		1

Figure 12: EE Reset Sequence

Figure 12 clearly shows this sequence being sent to the EEPROM and ensures its correctness.

Challenges

1 Challenge EE Write Timings

Figure 13 below shows the timing for an EEPROM write byte write. It took approximately 2.64 ms to complete. This byte write utilized ack polling, rather than a fixed delay, to measure exactly how long it took for a write cycle to complete. Note that the data sheet reflects a 5 ms max time. Figure 14 below shows the timing for an EEPROM page write of 16 bytes. It took approximately 11.80 ms.

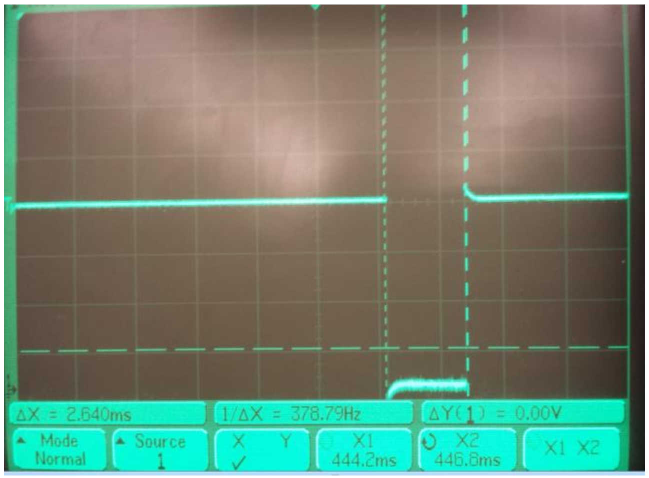


Figure 13: EE byte write timing

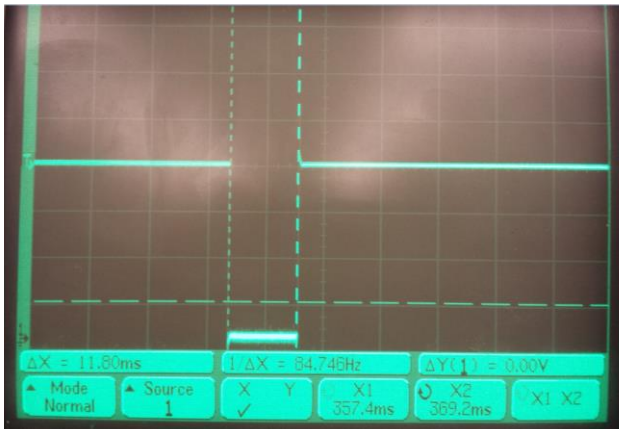


Figure 14: EE page write timing

The calculations below give a great perspective on how the performance of a page write is greater than the performance of a byte write. A byte write is forced to go through an entire write cycle after each byte, thus, 16 byte writes will include the overhead of 16 write cycles. A page write of 16 bytes only has the overhead of a single write cycle for 16 bytes. This reduces its execution time tremendously and this can be seen in figures 13 and 14 and the calculations below. In addition, the overhead for a byte write grows much faster than that of a page write, thus, a huge difference in performance can be seen calculating the time a byte write and page write of 1024 bytes.

$$Byte\ Write = 2.64\ ms\ per\ byte * 16\ bytes = 42.64\ ms$$

$$Page\ write\ (16\ bytes) = 11.80\ ms$$

$$Byte\ Write\ of\ 1024\ bytes = 2.64ms * 1024 = 2703.4\ ms$$

$$Page\ Write\ of\ 1024\ bytes = 11.80 * 64\ pages\ writes\ (16\ bytes\ each) = 755.2\ ms$$

Lab Questions

1. Windows 7
2. SDCC 2.6
3. Codeblocks 12.11
4. No
5. No